# Parcial de computabilidad

Lógica y computabilidad

Verano 2016

El examen es a libro abierto y se puede suponer demostrado lo dado en las clases y los ejercicios de las guías colocando referencias claras. Entregar cada ejercicio en hojas separadas. En cada hoja debe figurar nombre, apellido y número de orden. El examen consta de 4 ejercicios de igual valor. Cada ejercicio será calificado con A (aprobado), R (regular) o I (insuficiente), ocasionalmente con un signo - (menos). Para aprobar un parcial es necesario tener al menos dos ejercicios calificados con A o A-. Para promocionar es necesario tener al menos tres ejercicios calificados con A o A- en ambos parciales o sus correspondientes recuperatorios.

**Ejercicio 1.** Sea $ordenar : \mathbb{N} \to \mathbb{N}$ la función que ordena una lista de números naturales de menor a mayor. Más explícitamente, $ordenar(0) = 0$ y dado un número natural $x > 0$, lo interpreta como una lista de números naturales sin ceros al final y devuelve la codificación de la lista que resulta de ordenar los elementos de la original de menor a mayor. Demostrar que la función *ordenar* es primitiva recursiva.

**Ejercicio 2.** Sea $n \in \mathbb{N}$ un número natural fijo. Demostrar que la siguiente función es parcial computable:

$$f(y) = \begin{cases} 1 & \text{si existen } x_1, ..., x_n \mid \Phi_y^{(n)}(x_1, ..., x_n) \downarrow \text{ y } \Phi_y^{(n)}(x_1, ..., x_n) \neq ordenar([x_1, ..., x_n]) \\ \uparrow & \text{en caso contrario} \end{cases}$$

donde *ordenar* es la función del *Ejercicio 1* y puede asumirse el resultado de ese ejercicio como válido.

**Ejercicio 3.** Decidir si las siguientes funciones son computables o no. Justificar la respuesta.

a. $f(x, y) = \begin{cases} 1 & \text{si } \Phi_x^{(1)}(y) = 0 \text{ ó } \Phi_y^{(1)}(0) = 0 \\ 0 & \text{en caso contrario} \end{cases}$

b. $g(x) = \begin{cases} 1 & \text{si existe } y \geq x \mid \Phi_y^{(1)}(3x + 5) = 8 \\ 0 & \text{en caso contrario} \end{cases}$

**Ejercicio 4.** Sea $f : \mathbb{N} \to \mathbb{N}$ una función total, $G = \{\langle x, y\rangle : y = f(x)\}$ y $L = \{\langle x, y\rangle : y \leq f(x)\}$. Decidir si son verdaderas o falsas las siguientes afirmaciones. Justificar la respuesta.

a. Si $G$ es c.e. entonces $G$ es computable.

b. Si $L$ es c.e. entonces $L$ es computable.

| 1 | 2 | 3 | 4 | T |
|---|---|---|---|---|
| A | A | A | R | P |

N°orden: 24

9 (nueve)

## Ejercicio 1

En primer lugar, defino la función $posOrd : \mathbb{N}^2 \to \mathbb{N}$, que recibe una secuencia y un índice y devuelve el índice (actual) del elemento que habría en la posición solicitada si la ~~lista~~ secuencia estuviera ordenada de menor a mayor. Lo hago usando el esquema de recursión global (que se probó válido en el ej. (14) de la práctica 1).

$$posOrd(s,0) = f([\,], s)$$
$$posOrd(s,t+1) = f([posOrd(s,0), \ldots, posOrd(s,t)], s).$$

donde $f : \mathbb{N}^2 \to \mathbb{N}$ es la siguiente función p.r.:

$$f(ant, s) = \min_{i \leq |s|} \left(noEst\acute{a}(i, ant) \wedge (\forall j)_{\leq |s|} (noEst\acute{a}(j, ant) \Rightarrow s[i] \leq s[j])\right)$$

donde el predicado $noEst\acute{a} : \mathbb{N}^2 \to \{0,1\}$ es:

$$noEst\acute{a}(x,s) = (\forall i)_{\leq |s|} (s[i] \neq x)$$

que es claramente p.r.

Lo que hace $f$ es, conociendo los índices de los elementos anteriores ~~ellos~~ en la lista ordenada, buscar el menor índice que no sea uno de ellos y tal que todos los elementos cuyos índices tampoco están entre los "ya ordenados" sean mayores o iguales al elemento ocupado por dicho índice. En castellano: devuelvo el ~~el~~ índice del primer elemento de la ~~lista que~~ secuencia que no esté entre los que "ya ordené" y que sea menor o igual que todos los demás.

Una vez que tengo posOrd, puedo definir ordenar de manera primitiva recursiva como:

$$\text{ordenar}(s) = \prod_{i=1}^{|s|} \text{nprimo}(i)^{\left(s[\text{posOrd}(s,i)]\right)}$$

o, menos formalmente,

$$\text{ordenar}(s) = \left[ s[\text{posOrd}(s,1)],\ s[\text{posOrd}(s,2)],\ \ldots,\ s[\text{posOrd}(s,|s|)] \right]$$

(A)

## Ejercicio 2

Lo hago construyendo un programa que compute $f$:

```
1 [C] IF STP^(n)(Π_1^(n+1)(Z_1),..., Π_n^(n+1)(Z_1), X_1, Π_(n+1)^(n+1)(Z_1)) ≠ 0 GOTO A
2     Z_1 ← Z_1 + 1
3     GOTO C
4 [A] Z_2 ← r(SNAP^(n)(Π_1^(n+1)(Z_1),..., Π_n^(n+1)(Z_1), X, Π_(n+1)^(n+1)(Z_1))) [1]
5     Z_3 ← ordenar (sub(Z_1, 1, n))
6     IF Z_2 ≠ Z_3 GOTO B
7     Z_1 ← Z_1 + 1
8     GOTO C
9 [B] Y ← 1
```

Z1 no es una n-tupla?

~~Lo ideal es ordenar todos los posibles ...~~

Donde:

- Las funciones $\Pi_i^{(j)}$, para $i = 1, \dots, j$, representan a los observadores de $j$-tuplas. Es decir, $\Pi_i^{(j)}(x)$ devuelve el elemento en la $i$-ésima posición de $x$, mirando a $x$ como una $j$-tupla. Se demostró que estos observadores son p.r. en el ejercicio (12) de la práctica 1.
- En la línea 5 se usan "ordenar", que se asume p.r., y "sub", la función primitiva recursiva del ejercicio (13) de la práctica 1.

La idea del programa es similar a la del ejercicio (9) de la práctica 2. Se recorren todas las combinaciones posibles de valores para $(x_1, \dots, x_n, t)$, aprovechando la existencia de una biyección entre $\mathbb{N}$ y las $n+1$-tuplas de

naturales. El ciclo que forman las tres primeras líneas realiza esta tarea.

Si para algún conjunto de valores $x_1, \dots, x_n$ el programa termina, existirá un $t$ para el cual STP dé 1. La guarda del ciclo detecta esta situación y se pasa a la etiqueta [A], donde se verifica el resto de la condición que debe cumplirse para devolver 1: que $\Phi_y^{(n)}(x_1, \dots, x_n) \neq$ ordenar $([x_1, \dots, x_n])$.

El valor de $\Phi_y^{(n)}(x_1, \dots, x_n)$ se obtiene utilizando SNAP y se almacena en $Z_2$. La secuencia $[x_1, \dots, x_n]$ se obtiene como subsecuencia de $Z_1$ (que vale $[x_1, \dots, x_n, t]$)(*) y luego de llamar a "ordenar" con este valor, se almacena el resultado en $Z_3$. A continuación se comparan estos valores y, en caso de ser distintos, se devuelve 1. Si no, se intenta de nuevo con la secuencia siguiente.

(*) <u>NOTA</u>: La resolución tiene una inconsistencia, ya que $Z_1$ se interpreta a veces como secuencia y a veces como n+1-upla. Para corregir esto se puede reemplazar la línea 5 del programa por:

$$Z_3 \leftarrow \text{ordenar}\left(\prod_{i=1}^{n} \text{nprimo}(i)^{\left(\Pi_i^{(n+1)}(Z_1)\right)}\right)$$

es decir:

$$Z_3 \leftarrow \text{ordenar}\left(\left[\Pi_1^{(n+1)}(Z_1), \dots, \Pi_n^{(n+1)}(Z_1)\right]\right)$$

que es lo mismo que:

$$Z_3 \leftarrow \text{ordenar}\left([x_1, \dots, x_n]\right).$$

## Ejercicio 3

A (a) $f$ no es computable.

Demostración: Por el absurdo. Supongamos que $f$ fuera computable. Sea $f': \mathbb{N} \to \mathbb{N}$ la siguiente función:

$$f'(x) = f(x,x) = \begin{cases} 1 & \text{si } \phi_x^{(1)}(x) = 0 \vee \phi_x^{(1)}(0) = 0 \\ 0 & \text{en caso contrario} \end{cases}$$

Entonces $f'$ es claramente computable.

Sea además $g: \mathbb{N}^2 \to \mathbb{N}$ la siguiente función parcial-computable:

$$g(z,x) = \begin{cases} 0 & \text{si } \phi_x^{(1)}(x)\downarrow \\ \uparrow & \text{en caso contrario} \end{cases}$$

y sea $e \in \mathbb{N}$ tal que $\phi_e^{(2)}(z,x) = g(z,x)$.

Por el Teorema del Parámetro, existe $S: \mathbb{N}^2 \to \mathbb{N}$ p.r. tal que $\phi_{S(e,x)}^{(1)}(z) = g(z,x)$ $(\forall z)$.

Sea entonces $h(x) = f'(S(e,x))$. ¿Qué computa $h$?

i) Sea $x$ tal que $\phi_x^{(1)}(x)\downarrow$. Entonces:

$\phi_x^{(1)}(x)\downarrow \Rightarrow g(z,x) = 0$ $(\forall z)$ (por definición de $g$)

$\Rightarrow \phi_{S(e,x)}^{(1)}(z) = 0$ $(\forall z)$ (por definición de $S$)

$\Rightarrow \phi_{S(e,x)}^{(1)}(S(e,x)) = 0 \wedge \phi_{S(e,x)}^{(1)}(0) = 0$

$\Rightarrow \phi_{S(e,x)}^{(1)}(S(e,x)) = 0 \vee \phi_{S(e,x)}^{(1)}(0) = 0$

$\Rightarrow f'(S(e,x)) = 1$ (por definición de $f'$)

$\Rightarrow h(x) = 1$

ii) Consideremos el caso contrario:

$\phi_x^{(1)}(x)\uparrow \Rightarrow g(z,x)\uparrow$ $(\forall z)$ (por definición de $g$)

$\Rightarrow \phi_{S(e,x)}(z) \uparrow \quad (\forall z)$ (por definición de $S$)

$\Rightarrow \neg\left(\phi_{S(e,x)}(S(e,x)) = 0\right) \wedge \neg\left(\phi_{S(e,x)}(0) = 0\right)$

$\Rightarrow \neg\left(\phi_{S(e,x)}(S(e,x)) = 0 \vee \phi_{S(e,x)}(0) = 0\right)$

$\Rightarrow f'(S(e,x)) = 0$ (por definición de $f'$)

$\Rightarrow h(x) = 0$

En resumen:

$h(x) = \begin{cases} 1 & \text{si } \phi_x^{(1)}(x)\downarrow \\ 0 & \text{si no} \end{cases} = Halt(x,x)$. Pero dado que $h$ es la composición de $f'$ (computable) con $S$ (primitiva recursiva), $h$ debe ser computable. Esto es absurdo puesto que sabemos que $Halt(x,x)$ no es computable. Luego la suposición de que $f$ es computable debe ser errónea.

¡Qué complicado! Fijando $y =$ # de 1 x ej. y usando Rice salía mucho + sencillo :)

A (b) $g$ es computable. <u>Demostración</u>:

Sabemos que existen infinitos programas que calculan la constante 8. En particular, $(\forall x \in \mathbb{N})$ existen infinitos $y$ que cumplen $y \geq x$ y $\phi_y^{(1)}(z) = 8 \; (\forall z)$. Sea $y_0$ uno de estos valores. La función $h(x) = \phi_{y_0}^{(1)}(3x+5)$ es trivialmente computable (calcular $3x+5$ es p.r.) y vale que $h(x) = 8 \; (\forall x)$. En otras palabras, la condición para que $g(x) = 1$ vale siempre, siendo $y_0$ el $y$ buscado. Por lo tanto $g(x) = 1 \; (\forall x)$, es decir, $g$ es la función constante 1, que es computable (es primitiva recursiva).

Nº orden: 24

R

Ejercicio 4

ⓐ Verdadero. Demostración: Si G es c.e., es parcialmente computable la función

$$g(x,y) = \begin{cases} 1 & \text{si } \langle x,y\rangle \in G \\ \uparrow & \text{si no} \end{cases} = \begin{cases} 1 & \text{si } f(x)=y \\ \uparrow & \text{si no.} \end{cases}$$

Dado que f es total, $(\forall x)$ existe un (único) valor de y tal que $g(x,y)\downarrow$. (Este valor es $y=f(x)$).
Utilizando la idea de recorrer las duplas $\langle y,t\rangle$ mediante su biyección con los naturales y la función p.r. STP, es sencillo construir un programa que halle cuál es este valor de y, analizando para cada caso si, tras t pasos, terminó el cómputo de $g(x,y)$ (para algún programa que compute la función g).
Pero ahora, si se utiliza a este programa como una función de x (se toma a x como parámetro de entrada), se tiene ~~una función~~ un programa capaz de calcular $f(x)$ $(\forall x)$. Luego f es computable, por lo que puede verificarse de forma computable la validez de la condición de pertenencia a G, implicando que G es computable.